Path planning using intervals and graphs

Luc Jaulin

PATH PLANNING USING INTERVALS AND GRAPHS

Luc Jaulin

Laboratoire des Signaux et Systèmes, CNRS
Supélec, Plateau de Moulon,
91192 Gif-sur-Yvette Cedex, France.

On leave from
Laboratoire d'Ingénierie des Systèmes Automatisés,
ISTIA, 62 avenue Notre Dame du Lac,
49000 Angers, France.

Phone: (33) 1 69 85 17 59
Fax: (33) 1 69 41 30 60
Email: jaulin@lss.supelec.fr

Abstract: In this paper, the problem of interest is to find a path with given endpoints such that the path lies inside a compact set $\mathcal{S}$ given by nonlinear inequalities. The proposed approach uses interval analysis for characterizing $\mathcal{S}$ by subpavings (union of boxes) and graph algorithms for finding short feasible paths. As an illustration, the problem of finding collision-free paths for a polygonal rigid object through a space that is cluttered with segment obstacles is considered.

Keywords: Configuration space, Connexity, Graph, Interval analysis, Motion planning, Set inversion, Subpavings.

1. Introduction

In this paper, we present a new approach to find a collision-free path for an object in a given space with obstacles. The issue of path planning in a known environment has been addressed by many researchers (see, *e.g.,* [17], [14], [18] and [11]). Most of the current approaches to path planning are based on the concept of configuration space (C-space) [13]. Each coordinate of the C-space represents a degree of freedom of the object. The number of independent parameters needed to specify an object configuration corresponds to the

dimension of the C-space. The start configuration and the goal configuration become two points $\vec{a}$ and $\vec{b}$ of the C-space. An example of such objects are industrial robots which are kinematic chains in which adjacent links are connected by n prismatic or rotary joints, each with one degree of freedom. The positions and orientations of each link of the industrial robot can be characterized by n real numbers, which are the coordinates of a single n-dimensional point in the C-space (see [15], for more information).

The *feasible configuration space* $\mathcal{S}$ is the subset of the C-space corresponding to feasible configuration of the object, *i.e.*, $\mathcal{S}$ contains all configuration vectors for which the object does not collide with nearby obstacles. The path planning problem formulated in the C-space amounts to finding a path included in $\mathcal{S}$ from the start point $\vec{a}$ to the goal point $\vec{b}$. Many approaches to solve this problem are based on the use of potential functions, introduced by Khatib [10]. In the potential field approach, the obstacles to be avoided are represented by a repulsive potential, and the goal is represented by an attractive potential. According to the force generated by the sum of these potential fields, the object is expected to reach (if the method does not stop at any local minimum) its goal configuration without colliding with obstacles. Other approaches based on the subdivision of the C-space have also been considered (see [2], [1], [20]). They partition the C-space with a set of nonoverlapping boxes. Those that have been proved to be inside $\mathcal{S}$, those that have been proved to be outside $\mathcal{S}$, and those for which nothing has been proved. The existing methods used to decide if a box is inside or outside the feasible configuration space $\mathcal{S}$ are not based on interval analysis and are limited to a small class of problems and they meet some difficulties with orientation parameters.

Interval analysis is able to prove that a given box is inside or outside $\mathcal{S}$ for a huge class of problems and is used for the first time in this context (see also [8]) with classical methods based on the subdivision of the C-space. Note that interval analysis has already been used for parametric paths in [7] and [19], but the former methods require a parametric model for the path, *i.e.*, the path should be parametrized by a vector $\vec{p}$ to be tuned. This approach is limited to small-dimensional path models (*i.e.,* the dimension of $\vec{p}$ should be small). In [7] and also in [19], the model chosen for the path was a cubic polynomial. This paper presents a non parametric approach

Section 2 gives the basic notions for building a graph associated with the path planning problem. In Section 3, two algorithms able to find a feasible path from $\vec{a}$ to $\vec{b}$ are given. The first one characterizes $\mathcal{S}$ and then finds a feasible path. Except the fact that the tests used to decide about the feasibility of a box are based on interval analysis, this algorithm

is rather classical (see, *e.g.,* [2], [1]). The second algorithm, which is new and much more efficient than the first one, searches only the regions of the C-space that may lead to a good feasible path. As an application, the problem of moving a nonconvex polygonal object without colliding with segment obstacles is considered in Section 4.

2. Graph discretization of the configuration space

Interval analysis makes it possible to build powerful inclusion tests to prove that a box is inside or outside a set $\mathcal{S}$ given by nonlinear inequalities. Using a subdivision algorithm such as SIVIA (Set Inversion Via Interval Analysis see [6]), a guaranteed characterization of $\mathcal{S}$ can be obtained and then a graph associated with this characterization can be built. The whole procedure is called *graph discretization of* $\mathcal{S}$ and will be used in Section 3 to solve a path planning problem. This section gives the basic notions needed to understand the graph discretization principle.

2.1. Inclusion test

A *box* $[\vec{p}]$ of $\mathbb{R}^n$ is the Cartesian product of n intervals:

$$[\vec{p}] = \left[p_1^-, p_1^+\right] \times \cdots \times \left[p_n^-, p_n^+\right]. \tag{2.1}$$

The set of all boxes of $\mathbb{R}^n$ is denoted by $\mathbb{IR}^n$. The *width* of a box $[\vec{p}]$ is

$$\text{width}\left([\vec{p}]\right) = \max_{i\in\{1,\ldots,n\}} \left(p_i^+ - p_i^-\right) \tag{2.2}$$

A *Boolean number* is an element of $\mathbb{B} \triangleq \{0,1\}$ where 0 stands for false and 1 for true. By extension, a *Boolean interval* is an element of $\mathbb{IB} \triangleq \{\{0\},\{1\},\{0,1\}\}$. For simplicity and for similarity with classical interval notations, $\{0\},\{1\}$ and $\{0,1\}$ will be denoted by $0,1$ and $[0,1]$, respectively. If a is a Boolean interval,

$$0.a = 0;\ 1.a = a;\ 0 + a = a;\ 1 + a = 1;\ a.a = a + a = a. \tag{2.3}$$

For example, we have

$$\left([0,1] + 1\right).\left([0,1].1\right) = 1.[0,1] = [0,1].$$

An *inclusion test* for the Boolean function (or *test*) $t : \mathbb{R}^n \rightarrow \{0,1\}$ is a function $[t] : \mathbb{IR}^n \rightarrow \mathbb{IB}$ such that for all boxes $[\vec{p}] \in \mathbb{IR}^n$,

$$\begin{array}{rcl} [t]\left([\vec{p}]\right) = 1 & \Rightarrow & \forall \vec{p} \in [\vec{p}]\ ,\ t(\vec{p}) = 1, \\ [t]\left([\vec{p}]\right) = 0 & \Rightarrow & \forall \vec{p} \in [\vec{p}]\ ,\ t(\vec{p}) = 0. \end{array} \tag{2.4}$$

An inclusion test $[t]$ is *thin* if, for all vectors $\vec{p}$, $[t]([\vec{p}]) = t(\vec{p})$. $[t]$ is *minimal* if

$$\forall [\vec{p}] \in \mathbb{IR}^n, [t]([\vec{p}]) = \{t(\vec{p}), \vec{p} \in [\vec{p}]\}. \tag{2.5}$$

Example 1: Consider the test

$$t : \begin{array}{ccc} \mathbb{R}^2 & \rightarrow & \{0,1\} \\ (p_1,p_2)^{\mathrm{T}} & \rightarrow & (p_1 = 5), \end{array} \tag{2.6}$$

i.e., $t(\vec{p}) = 1$ if and only if $p_1 = 5$. The minimal inclusion test $[t]$ is given by

$$[t]([\vec{p}]) = \begin{cases} 1 & \text{if} \quad [p_1] = 5 \\ 0 & \text{if} \quad 5 \notin [p_1] \\ [0,1] & \text{otherwise} \end{cases} \tag{2.7}$$

Interval analysis (see, *e.g.*, [16]) makes it possible to build thin inclusion tests for a large class of tests involving nonlinear constraints. The inclusion tests will be used for our path planning problem to prove that a given box $[\vec{p}]$ is inside or outside the feasible configuration space $\mathcal{S}$.

2.2. Pavings

A *paving* of a box $[\vec{p}_0]$ is a set of nonoverlapping boxes, the union of which is equal to $[\vec{p}_0]$. On Figure 1, a paving $\mathcal{P} = \{[\vec{p}_1], [\vec{p}_2], \ldots, [\vec{p}_9]\}$ of the box $[\vec{p}_0] = [-2,10] \times [-2,6]$ is represented.

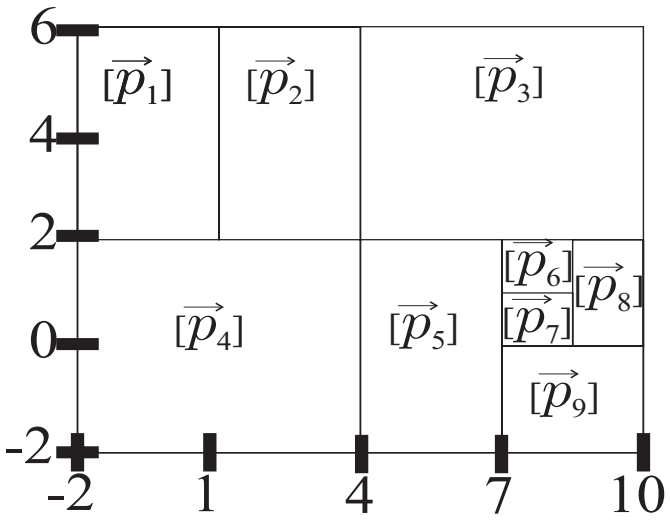

Figure 1: A paving with 9 boxes

Two boxes are *neighbors* if they have two overlapping faces. For instance, $[\vec{p}_1]$ and $[\vec{p}_4]$ are neighbors, but $[\vec{p}_2]$ and $[\vec{p}_5]$ are not neighbors. A *subbox* of $\mathcal{P}$ is an element of $\mathcal{P}$. For instance, $[\vec{p}_6]$ is a subbox of $\mathcal{P}$, but $[-2,4]\times[-2,6]$ is not a subbox of $\mathcal{P}$. A *subpaving* of $\mathcal{P}$ is a set nonoverlapping subboxes of $\mathcal{P}$. For instance, $\{[\vec{p}_6],[\vec{p}_7]\}$ is a subpaving of $\mathcal{P}$. The subpaving $\mathcal{P}_1$ of a paving $\mathcal{P}$ that contains all boxes of $\mathcal{P}$ that satisfy a given condition is denoted by

$$\mathcal{P}_1 = \text{ Subpaving}\left(\mathcal{P}, \text{Condition}\left([\vec{p}]\right)\right) \tag{2.8}$$

Consider for instance the test $t(\vec{p}) \triangleq (p_1 = 5)$ where p_1 is the first component of $\vec{p}$. If $[t]\left([\vec{p}]\right)$ is the minimal inclusion test for $t(\vec{p})$ as defined by (2.7), since $[t]\left([\vec{p}]_3\right) = [t]\left([\vec{p}_5]\right) = [0,1]$

$$\text{Subpaving}\left(\mathcal{P}, [t]\left([\vec{p}]\right) = 0\right) = \{[\vec{p}_1],[\vec{p}_2],[\vec{p}_4],[\vec{p}_6],[\vec{p}_7],[\vec{p}_8],[\vec{p}_9]\} \tag{2.9}$$

2.3. Graphs

In this subsection, basic definitions related to graphs are given. For a detailed introduction to the fundamentals of graph theory and its varied uses in many fields of modern technology, see, *e.g.,* the book of Deo [3]. A *graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of a non-empty set $\mathcal{V}$ of vertices and a set $\mathcal{E}$ of unordered pairs of vertices of $\mathcal{V}$ called *edges*. If v_a and v_b are two vertices of the graph, an edge associated with the pair (v_a, v_b) is denoted by ab.

A *walk* in $\mathcal{G}$ is a sequence of k vertices $(v_1, \ldots, v_k)$ such that for all $i \in (1, \ldots, k-1)$, the edge $v_i v_{i+1}$ belongs to $\mathcal{E}$. The walk is a *path* if $v_i \neq v_j$ for $i \neq j$. The walk is a *cycle* if $v_k = v_1$. A graph is *connected* if there is a path between any pair of vertices. Two vertices v_i and v_j of $\mathcal{G}$ are *neighbors* if the edge $v_i v_j$ exists in $\mathcal{E}$. A *subgraph* of $\mathcal{G}$ is a graph whose vertices and edges belong to $\mathcal{G}$.

Any paving or subpaving $\mathcal{P}$ of a box $[\vec{p}_0]$ can be represented by a graph $\mathcal{G}$. Each subbox $[\vec{p}_i]$ of $\mathcal{P}$ is associated with a vertex v_i of $\mathcal{G}$. If two boxes $[\vec{p}_i]$ and $[\vec{p}_j]$ are neighbors in $\mathcal{P}$, then the edge $v_i v_j$ exists in $\mathcal{G}$. For instance, the graph $\mathcal{G}$ associated with the paving of Figure 1 is given on Figure 2 and the graph $\mathcal{G}_1$ associated with the subpaving (2.9) is given on Figure 3.

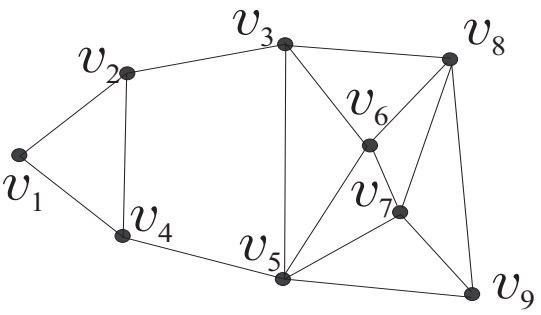

Figure 2: The graph $\mathcal{G}$ associated with the paving of Figure 1

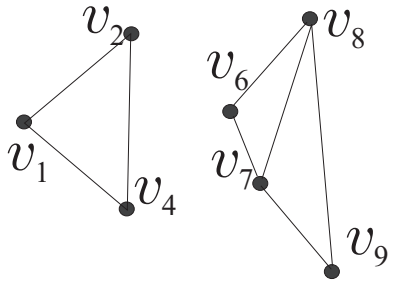

Figure 3: Graph $\mathcal{G}_1$ associated the subpaving (2.9).
G_1 is disconnected and is a subgraph of $\mathcal{G}$.

Note that the graph $\mathcal{G}_1$ is not connected. It is a subgraph of $\mathcal{G}$. The graph associated with a paving (or subpaving) $\mathcal{P}$ is denoted by $\mathcal{G} = \text{graph}(\mathcal{P})$.

3. Algorithms for finding a feasible path

Consider a compact set $\mathcal{S}$ included in a box $[\vec{p}_0]$ and two points $\vec{a}$ and $\vec{b}$ of $\mathcal{S}$. Assume that a thin inclusion test $[t]$ is available to decide if a box is inside or outside $\mathcal{S}$. A *motion* from the start point $\vec{a}$ to the goal point $\vec{b}$ is a one-to-one continuous function $\vec{m} : [0,1] \to \mathcal{R}^n; t \to \vec{m}(t)$, such that $\vec{m}(0) = \vec{a}$ and $\vec{m}(1) = \vec{b}$. The associated path is the set $\ell = \{\vec{m}(t) | t \in [0,1]\}$. The path ℓ is *feasible,* if $\ell \subset \mathcal{S}$. In this section, two algorithms FeasiblePath1 and FeasiblePath2 are given. Both return a *box path, i.e.,* a list of adjacent boxes $\{[\vec{p}_a], [\vec{p}_1], \ldots, [\vec{p}_{\ell-1}], [\vec{p}_b]\}$, $\vec{p}_a \in [\vec{p}_a], \vec{p}_b \in [\vec{p}_b]$, such that all these boxes are inside $\mathcal{S}$. When such a box path is found, it is still necessary to find a feasible point path ℓ from $\vec{a}$ to $\vec{b}$ of the moving object through the configuration space. In general, the choice of the final point path should be based on domain-specific considerations such as kinematic or dynamic characteristics, not on purely geometric criteria [12]. For instance, a desirable property of the final path is that it be smooth. Here, for the sake of simplicity,

a broken line from $\vec{a}$ to $\vec{b}$ lying inside the box path is chosen.

At the beginning of this section, an algorithm able to find the shortest path in a graph is recalled. It will be used as a subroutine by the two algorithms FeasiblePath1 and FeasiblePath2.

3.1. Finding the shortest path in a graph

Among the algorithms that have been proposed for the shortest path between two specified vertices v_a and v_b in a graph $\mathcal{G}$, perhaps the most efficient one is an algorithm due to Dijkstra [4]. Although it has been initially given for weighted *digraphs* (graphs with directed edges), here a simplified version is presented in Table 1 for (not directed) graphs. To each vertex v of $\mathcal{G}$ is associated an integer $d(v)$ representing the minimum number of edges between vertices v_a and v. $\mathcal{G}(i), i \in \mathcal{N}$ denotes the set of all vertices of $\mathcal{G}$ such that $d(v) = i$. If the algorithm ShortestPath returns "Failure", then v_a and v_b are not in the same connected component of $\mathcal{G}$. Otherwise, it returns a shortest path in the graph.

ShortestPath$(\mathcal{G}, v_a, v_b)$
For each vertex $v \in \mathcal{G}$, set $d(v) = \infty$;
$d(v_a) = 0$; $d_{\text{mini}} = 0$;
Repeat
If $\mathcal{G}(d_{\text{mini}}) = \emptyset$ return ("Failure");
$d_{\text{mini}} = d_{\text{mini}} + 1$;
For each vertex $v \in \mathcal{G}(d_{\text{mini}} - 1)$;
For each neighbors w of v in $\mathcal{G}$ with $d(w) = \infty$, $d(w) = d_{\text{mini}}$;
Until $d(v_b) \neq \infty$;
$\ell = d(v_b)$; $v_\ell = v_b$;
For $i = \ell - 1$ down to 0, select a neighbor v_i of v_{i+1} such that $d(v_i) = i$;
Return $(\{v_a, v_1, v_2, \ldots, v_{\ell-1}, v_b\})$.

Table 1: Algorithm ShortestPath

Run ShortestPath$(\mathcal{G}, v_1, v_6)$ with the graph of Figure 2. We get $d(v_1) = 0, d(v_2) = d(v_4) = 1, d(v_3) = d(v_5) = 2, d(v_6) = d(v_7) = d(v_8) = d(v_9) = 3$. ShortestPath returns the path $\{v_1, v_2, v_3, v_6\}$ or the path $\{v_1, v_4, v_5, v_6\}$.

3.2. Algorithm FeasiblePath1

The first part of the algorithm FeasiblePath1, given in Table 2, is the procedure Sivia (for Set Inversion Via Interval Analysis) (see [6]). Sivia builds two subpavings: $\mathcal{P}^-$ and $\mathcal{P}^+$ satisfying $\mathcal{P}^- \subset \mathcal{S} \subset \mathcal{P}^+$ and uses a stack of boxes to store all boxes still to be studied. The graphs $\mathcal{G}^-$ and $\mathcal{G}^+$ associated with $\mathcal{P}^-$ and $\mathcal{P}^+$ are then built. Then, the algorithm selects two boxes $[\vec{p}_a]$ and $[\vec{p}_b]$ of $\mathcal{P}^+$ such that $\vec{a} \in [\vec{p}_a]$ and $\vec{b} \in [\vec{p}_b]$. Note that two or more acceptable candidates $[\vec{p}_a]$ and $[\vec{p}_b]$ may exist if $\vec{a}$ or $\vec{b}$ is on the boundary of a box of $\mathcal{P}^+$. In such a case, the algorithm selects the first one it has found. Denote by v_a and v_b the two vertices of $\mathcal{G}^+$ associated with $[\vec{p}_a]$ and $[\vec{p}_b]$. FeasiblePath1 calls the procedure ShortestPath to get a path $\mathcal{L}^+$ of $\mathcal{G}^+$ from v_a to v_b. If no path is found, no path exists in $\mathcal{P}^+$ from $\vec{a}$ to $\vec{b}$ and FeasiblePath1 returns ("No path"). In such a case, $\vec{a}$ and $\vec{b}$ are proved to belong to two different connected components of $\mathcal{S}$[1]. If a nonempty $\mathcal{L}^+$ is found, ShortestPath is run again to find a path $\mathcal{L}^-$ of $\mathcal{G}^-$ that links v_a to v_b. If a non empty path $\mathcal{L}^- = \{v_a, v_1, \ldots, v_{\ell-1}, v_b\}$ is found, the associated box path in $\mathcal{P}^-$ is included in $\mathcal{S}$ and a point path can thus be generated. If $\mathcal{L}^-$ is empty, the algorithm returns "Failure" because nothing has been proved about the existence of a feasible path from $\vec{a}$ to $\vec{b}$. One can try to run again the algorithm with a smaller ε.

[1] **Explanation**: If $\mathcal{L}^+ = \emptyset$, then the two vertices v_a and v_b belong to two distinct connected component of the graph $\mathcal{G}^+$, *i.e.*, the two points $\vec{a}$ and $\vec{b}$ belong to two distinct connected component of the subpaving $\mathcal{P}^+$. Now, since $\vec{a} \in \mathcal{S}, \vec{b} \in \mathcal{S}$ and $\mathcal{S} \subset \mathcal{P}^+$ then $\vec{a}$ and $\vec{b}$ belong to two distinct connected component of $\mathcal{S}$. Thus, when FeasiblePath1 returns ("No path"), the conclusion is obtained in a guaranteed way (provided that outward rounding has been implemented for intervals).

FeasiblePath1$\left([t]\,,\vec{a},\vec{b},[\vec{p}_0]\,,\varepsilon\right)$

If $[t]\,(\vec{a}) \neq 1$ or $[t]\,(\vec{b}) \neq 1$, return ("Error: $\vec{a}$ and $\vec{b}$ should be feasible");

If $\vec{a} \notin [\vec{p}_0]$ or $\vec{b} \notin [\vec{p}_0]$, return ("Error: $\vec{a}$ and $\vec{b}$ should belong to $[\vec{p}_0]$");

Stack $= \{[\vec{p}_0]\}$; $\Delta\mathcal{P} = \emptyset$; $\mathcal{P}^- = \emptyset$;

While Stack $\neq \emptyset$;

  Pop into $[\vec{p}]$;

  If $[t]([\vec{p}]) = 1$, $\mathcal{P}^- = \mathcal{P}^- \cup \{[\vec{p}]\}$;

  If $[t]([\vec{p}]) = [0,1]$ and width$([\vec{p}]) \leq \varepsilon$, $\Delta\mathcal{P} = \Delta\mathcal{P} \cup \{[\vec{p}]\}$;

  If $[t]([\vec{p}]) = [0,1]$ and width$([\vec{p}]) > \varepsilon$,

    Bisect$([\vec{p}])$ and stack the two resulting boxes;

EndWhile;

$\mathcal{P}^+ = \mathcal{P}^- \cup \Delta\mathcal{P}$; $\mathcal{G}^+ = \text{Graph}(\mathcal{P}^+)$; $\mathcal{G}^- = \text{Graph}(\mathcal{P}^-)$;

$v_a = \text{vertex}([\vec{p}_a])$, where $[\vec{p}_a] \in \mathcal{P}^+$ and $\vec{a} \in [\vec{p}_a]$;

$v_b = \text{vertex}([\vec{p}_b])$, where $[\vec{p}_b] \in \mathcal{P}^+$ and $\vec{b} \in [\vec{p}_b]$;

$\mathcal{L}^+ =$ ShortestPath $(\mathcal{G}^+, v_a, v_b)$; If $\mathcal{L}^+ = \emptyset$, return ("No path");

If $v_a \notin \mathcal{G}^-$ or $v_b \notin \mathcal{G}^-$, return ("Failure");

$\mathcal{L}^- =$ ShortestPath $(\mathcal{G}^-, v_a, v_b)$; If $\mathcal{L}^- \neq \emptyset$, return $\mathcal{L}^-$ else return ("Failure");

Table 2: algorithm FeasiblePath1

3.3. Algorithm FeasiblePath2

The motivation of the new algorithm FeasiblePath2, presented in Table 3, is that for many path planning problems, the computing time of the graph algorithms are low compare to that of evaluating the inclusion tests $[t]$. FeasiblePath2 presented in Table 3 chooses carefully the boxes to be bisected. It first finds a shortest path $\mathcal{L}^+$ in the graph associated with an available outer subpaving $\mathcal{P}^+$ of $\mathcal{S}$. If no path is found, $\vec{a}$ and $\vec{b}$ are not in the same connected component of $\mathcal{S}$ and the algorithm returns ("No path"). If the path exists, then FeasiblePath2 tries to find the shortest path $\mathcal{L}^-$ in the graph $\mathcal{G}^-$ associated with an inner subpaving $\mathcal{P}^-$ of $\mathcal{S}$. If a path is found, it is returned. Otherwise, the box path corresponding to $\mathcal{L}^+$ has a good chance to contain a feasible path. Thus all subboxes of this path are bisected and new subpavings $\mathcal{P}^-$ and $\mathcal{P}^+$ are thus obtained. The whole procedure is performed again until a conclusion is reached.

FeasiblePath2$\left([t], \vec{a}, \vec{b}, [\vec{p}_0]\right)$

If $[t](\vec{a}) \neq 1$ or $[t](\vec{b}) \neq 1$, return ("Error: $\vec{a}$ and $\vec{b}$ should be feasible");

If $\vec{a} \notin [\vec{p}_0]$ or $\vec{b} \notin [\vec{p}_0]$, return ("Error: $\vec{a}$ and $\vec{b}$ should belong to $[\vec{p}_0]$");

Denote by $\mathcal{P}$ the paving containing the single box $[\vec{p}_0]$;

Repeat

$\mathcal{P}^+ =$ Subpaving($\mathcal{P}$, $1 \in [t]([\vec{p}])$); $\mathcal{G}^+ =$ Graph($\mathcal{P}^+$);

$v_a =$ vertex($[\vec{p}_a]$), where $[\vec{p}_a] \in \mathcal{P}^+$ and $\vec{a} \in [\vec{p}_a]$;

$v_b =$ vertex($[\vec{p}_b]$), where $[\vec{p}_b] \in \mathcal{P}^+$ and $\vec{b} \in [\vec{p}_b]$;

$\mathcal{L}^+ =$ ShortestPath $(\mathcal{G}^+, v_a, v_b)$;

If $\mathcal{L}^+ = \emptyset$, return ("No path");

$\mathcal{P}^- =$ Subpaving($\mathcal{P}$,$[t]([\vec{p}]) = 1$); $\mathcal{G}^- =$ Graph($\mathcal{P}^-$);

If $v_a \in \mathcal{G}^-$ and $v_b \in \mathcal{G}^-$, $\mathcal{L}^- =$ ShortestPath $(\mathcal{G}^-, v_a, v_b)$;

If $\mathcal{L}^- \neq \emptyset$, return $\mathcal{L}^-$;

$\mathcal{C} = \{[\vec{p}] \in \mathcal{P}^+ |$ vertex$([\vec{p}]) \in \mathcal{L}^+$ and $[t]([\vec{p}]) = [0,1]\}$;

Bisect all subboxes of $\mathcal{C}$, thus obtaining a new paving $\mathcal{P}$;

Until False.

Table 3: the new algorithm FeasiblePath2

4. Example

In this section, a new test case will be presented and solved using FeasiblePath1 and FeasiblePath2. The test-case presents two main advantages: *(i)* the C-space is two-dimensional so that pictures of $\mathcal{S}$ can be provided to illustrate the principle of the algorithms and *(ii)* the motion from the initial configuration to the goal configuration is not so easy to find by hand.

Consider a 2-dimensional room which contains $j_{\max} = 2$ segment obstacles. The extreme points of the jth segment are denoted $\vec{a}_j$ and $\vec{b}_j$ for $j \in \mathcal{J} = \{1, \ldots, j_{\max}\}$. The object to be moved is a nonconvex polygon with $i_{\max} = 14$ vertices, denoted by $\vec{s}_i \in \mathbb{R}^2, i \in \mathcal{I} = \{1, \ldots, i_{\max}\}$. The first vertex $\vec{s}_1$ is constrained to stay on the horizontal line with equation $y = 0$ in the room frame. The configuration of the object is thus represented by a two dimensional vector $\vec{p} = (p_1, p_2)^{\mathrm{T}}$, where p_1 is x-coordinate of $\vec{s}_1$ in the room frame and p_2 is the heading angle (in radian) of the object with respect to the a horizontal direction

to the right. In the object frame, the x-coordinates of the $\vec{s}_i$'s are given by

$$\{0, 0, 14, 14, 10, 10, 12, 12, 2, 2, 18, 18, 20, 20\}, \tag{4.1}$$

and the y-coordinates by

$$\{0, 14, 14, 6, 6, 8, 8, 12, 12, 2, 2, 18, 18, 0\}. \tag{4.2}$$

The coordinates of the segment obstacles in the room frame are given by

$$\vec{a}_1 = (8, 10)\,; \vec{b}_1 = (11, 10)\,; \vec{a}_2 = (25, 10)\,; \vec{b}_2 = (28, 10)\,. \tag{4.3}$$

Figure 4 illustrates the notion of configuration space for our test-case.

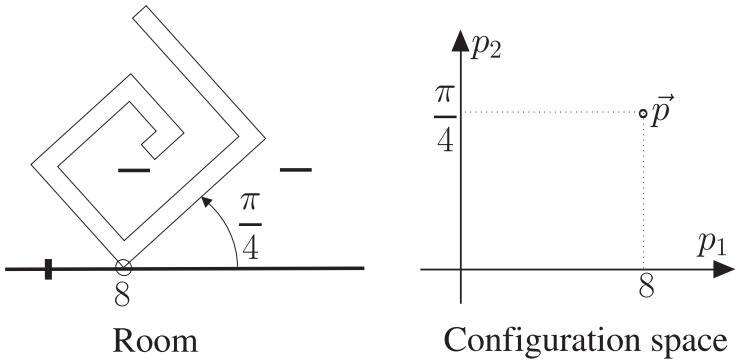

Figure 4: To a given configuration of the object in the room is associated a single point $\vec{p}$ in the C-space.

Figure 5 represents the initial configuration $\vec{p} = (0, 0)^{\mathrm{T}}$ and the goal configuration $\vec{p} = (17, 0)^{\mathrm{T}}$ of the object, respectively.

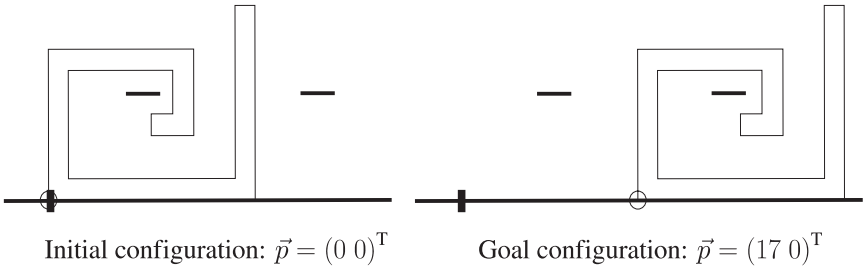

Figure 5: Initial and goal configurations for the object

A vector $\vec{p}$ associated with a given configuration for the object is *feasible* if and only if

- none of the object's edges intersects one of the segment obstacles.

- all extreme points of each segment obstacles are outside the object.

Note that, as illustrated in Figure 4, the vector $\vec{p} = (8,\ \pi/4)^{\mathrm{T}}$ is feasible. Denote by $\mathcal{S}$ the set of all feasible configuration vector $\vec{p}$ for the object. In what follows, if $\vec{a}$ and $\vec{b}$ are two points of $\mathcal{R}^2$, $\left[\vec{a},\vec{b}\right]$ denotes the segment with endpoints $\vec{a}$ and $\vec{b}$, and $\left(\vec{a},\vec{b}\right)$ is the line supported by $\vec{a}$ and $\vec{b}$. If $\mathcal{A}$ is a compact set, the smallest box which contains $\mathcal{A}$ is denoted by $[\mathcal{A}]$. For instance $\left[\vec{a} \cup \vec{b}\right]$ is the smallest box that contains both $\vec{a}$ and $\vec{b}$. We set $\vec{s}_{i_{\max}+1} := \vec{s}_1$. We have

$$\vec{p} \in \mathcal{S} \Leftrightarrow \begin{cases} \forall i \in \mathcal{I}, \forall j \in \mathcal{J}, [\vec{s}_i, \vec{s}_{i+1}] \cap \left[\vec{a}_j, \vec{b}_j\right] = \emptyset \text{ and} \\ \vec{a}_j \text{ and } \vec{b}_j \text{ are outside the object} \end{cases} \tag{4.4}$$

To test if $[\vec{s}_i, \vec{s}_{i+1}] \cap \left[\vec{a}_j, \vec{b}_j\right] = \emptyset$, we use the following equivalence

$$[\vec{s}_i, \vec{s}_{i+1}] \cap \left[\vec{a}_j, \vec{b}_j\right] \neq \emptyset \Leftrightarrow \begin{cases} (\vec{s}_i, \vec{s}_{i+1}) \cap \left[\vec{a}_j, \vec{b}_j\right] & \neq \emptyset \text{ and} \\ [\vec{s}_i, \vec{s}_{i+1}] \cap \left(\vec{a}_j, \vec{b}_j\right) & \neq \emptyset \text{ and} \\ [\vec{s}_{i+1} \cup \vec{s}_i] \cap \left[\vec{a}_j \cup \vec{b}_j\right] & \neq \emptyset \end{cases} \tag{4.5}$$

The last condition is important for the degenerated situation where the points $\vec{s}_{i+1}, \vec{s}_i, \vec{a}_j$ and $\vec{b}_j$ are aligned. In such a case, the two first conditions are always fulfilled and only the last condition is useful.

The algorithm of Table 4 decides if the configuration vector $\vec{p}$ is feasible. For a given segment number j, $\tilde{a} = (\tilde{x}_a, \tilde{y}_a)^{\mathrm{T}}$ and $\tilde{b} = (\tilde{x}_b, \tilde{y}_b)^{\mathrm{T}}$ represent the extreme points of the jth segment $\vec{a}$ and $\vec{b}$ in the object frame. The four first statements of the *j-for* loop compute the coordinates of the jth segment obstacle in the object frame. To prove that $\tilde{a}$ is inside the object, it suffices to check that $\sum_{i=1}^{i_{\max}} \arg(\vec{s}_i - \tilde{a}, \vec{s}_{i+1} - \tilde{a}) \neq 0$. The same can be done for $\tilde{b}$. If $d_1 \leq 0$ and $d_2 \leq 0$ and $\left[\tilde{a} \cup \tilde{b}\right] \cap [\vec{s}_{i+1} \cup \vec{s}_i] \neq \emptyset$, then, because of formula (4.5) the ith object's edge intersects the jth segment obstacle and thus the configuration $\vec{p}$ is unfeasible.

$t(\vec{p})$

For $j = 1$ to $j_{\max}$,

$\tilde{x}_a = (x_a(j) - p_1)\cos p_2 + y_a(j)\sin p_2$;

$\tilde{y}_a = -(x_a(j) - p_1)\sin p_2 + y_a(j)\cos p_2$;

$\tilde{x}_b = (x_b(j) - p_1)\cos p_2 + y_a(j)\sin p_2$;

$\tilde{y}_b = -(x_a(j) - p_1)\sin p_2 + y_a(j)\cos p_2$;

$\tilde{a} = (\tilde{x}_a, \tilde{y}_a)^{\mathrm{T}}$; $\tilde{b} = (\tilde{x}_b, \tilde{y}_b)^{\mathrm{T}}$;

If $\tilde{a}$ is inside the object, return 0;

If $\tilde{b}$ is inside the object, return 0;

For $i = 1$ to $i_{\max}$,

$d_1 = \det\left(\vec{s}_i - \tilde{b}, \vec{s}_i - \tilde{a}\right) * \det\left(\vec{s}_{i+1} - \tilde{b}, \vec{s}_{i+1} - \tilde{a}\right)$;

$d_2 = \det(\vec{s}_{i+1} - \vec{s}_i, \vec{s}_i - \tilde{a}) * \det\left(\vec{s}_{i+1} - \vec{s}_i, \vec{s}_i - \tilde{b}\right)$;

if $d_1 \leq 0$ and $d_2 \leq 0$ and $\left[\tilde{a} \cup \tilde{b}\right] \cap [\vec{s}_{i+1} \cup \vec{s}_i] \neq \emptyset$, return 0;

EndFor;

EndFor;

Return 1;

Table 4: Algorithm to test if a configuration vector $\vec{p}$ is feasible

An inclusion test $[t]([\vec{p}])$ for $t(\vec{p})$ is given by the algorithm of Table 5. To evaluate $[\tilde{x}_a]$, $[\tilde{y}_a]$, $[\tilde{x}_b]$, $[\tilde{y}_b]$, $[\tilde{x}_b]$, $[d_1]$, $[d_2]$, the centered form has been used with respect to p_1 and p_2. Procedures to prove that the boxes $[\tilde{a}]$ and $[\tilde{b}]$ are inside the polygonal object can be found in [9].

$[t]\,([\vec{p}])$

result = 1;

For $j = 1$ to $j_{\max}$,

$[\tilde{x}_a] = (x_a(j) - p_1)\cos[p_2] + y_a(j)\sin[p_2]$;

$[\tilde{y}_a] = -(x_a(j) - p_1)\sin[p_2] + y_a(j)\cos[p_2]$;

$[\tilde{x}_b] = (x_b(j) - p_1)\cos[p_2] + y_a(j)\sin[p_2]$;

$[\tilde{y}_b] = -(x_a(j) - p_1)\sin[p_2] + y_a(j)\cos[p_2]$;

$[\tilde{a}] = ([\tilde{x}_a], [\tilde{y}_a])^{\mathrm{T}}$; $[\tilde{b}] = ([\tilde{x}_b], [\tilde{y}_b])^{\mathrm{T}}$;

If $\Big([\tilde{a}]$ or $[\tilde{b}]$ are inside the object$\Big)$, return 0;

For $i = 1$ to $i_{\max}$,

$[d_1] = \det\Big(\vec{s}_i - [\tilde{b}], \vec{s}_i - [\tilde{a}]\Big) * \det\Big(\vec{s}_{i+1} - [\tilde{b}], \vec{s}_{i+1} - [\tilde{a}]\Big)$;

$[d_2] = \det(\vec{s}_{i+1} - \vec{s}_i, \vec{s}_i - [\tilde{a}]) * \det\Big(\vec{s}_{i+1} - \vec{s}_i, \vec{s}_i - [\tilde{b}]\Big)$;

if $([d_1] < 0$ and $[d_2] < 0)$ return 0;

if $(0 \in [d_1]$ or $0 \in [d_2])$ and $\Big[[\tilde{a}] \cup [\tilde{b}]\Big] \cap [\vec{s}_{i+1} \cup \vec{s}_i] \neq \emptyset$

result = $[0, 1]$;

EndFor;

EndFor;

Return result;

Table 5: Inclusion test algorithm.

In less than 10 minutes on a Pentium 133 and for $\varepsilon = 0.1$, FeasiblePath1 generates the paving presented on Figure 6, with the associated graph. Grey boxes are proved to be feasible and the black boxes are proved unfeasible. In less than 0.1 seconds, FeasiblePath1 finds the shortest path in the graph. The corresponding motion is displayed on Figure 7 and as expected, the two obstacle segments still appear on the figure. For $\varepsilon = 0.2$, FeasiblePath1 fails and is unable to find a feasible path.

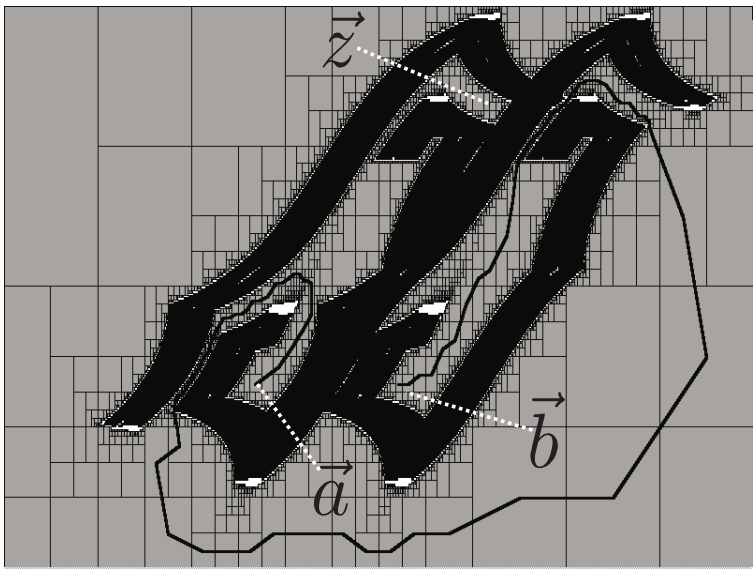

Figure 6: Paving and path generated by FeasiblePath1.
The frame corresponds to the search box $[\vec{p}_0] = [-28, 57] \times [-1.4, 2.7]$.

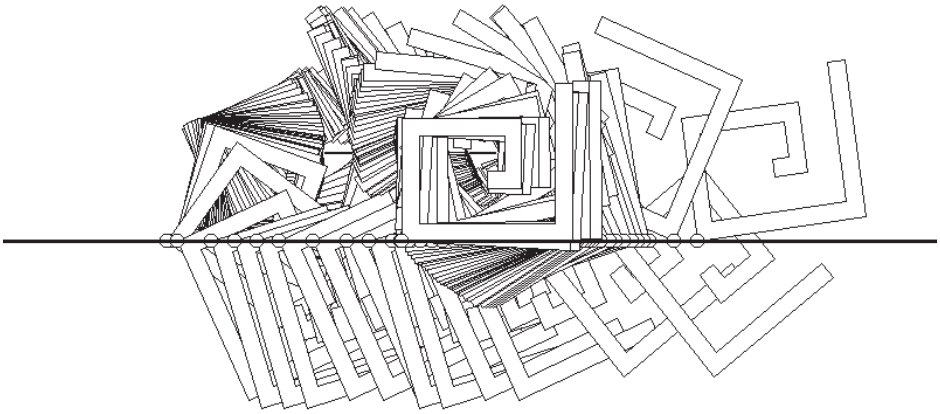

Figure 7: Display of the motion. The two segment obstacles are still visible

The configuration represented on Figure 8 and corresponding to the vector $\vec{z}$ of Figure 7 is a natural deadlock if one tries to solve the problem by hand.

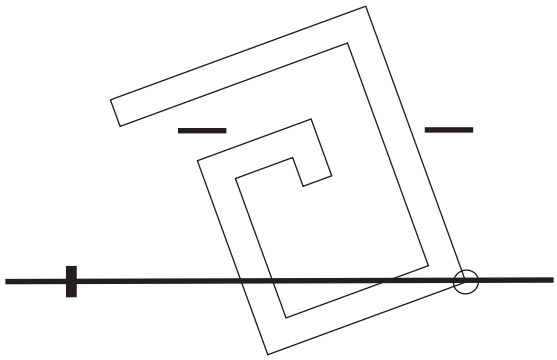

Figure 8: A natural deadlock if one tries to solve the problem by hand

In less than 1 minute on a Pentium 133, FeasiblePath2 finds the path shown on Figure 9. Grey boxes are proved to be inside $\mathcal{S}$, black boxes are outside $\mathcal{S}$ and nothing is known about white boxes. Note that FeasiblePath2 puts efforts to bisect and analyze zones of the C-space only when needed.

Figure 9: Paving and path generated by FeasiblePath2.
The frame corresponds to the search box $[\vec{p}_0] = [-28, 57] \times [-1.4, 2.7]$.

Remark: The frame box $[\vec{p}_0] = [-28, 57] \times [-1.4, 2.7]$ has been chosen small enough to make visible the small boxes and large enough to include the whole path. The example has also been tested for $[\vec{p}_0] = [-100, 100] \times [-10, 10]$. The computing time obtained by

FeasiblePath1 is about three times larger than for the former $[\vec{p}_0]$ whereas the computing time obtained by FeasiblePath2 remains unchanged. ♢

5. Conclusion

In this paper, a configuration-space approach has been used to solve the path planning problem. Interval analysis is used for the first time in this context to prove whether a box of the configuration space is feasible or not. Combining interval tools and graph tools, two algorithms have been presented to find a good feasible path from a start configuration to a goal configuration. As an application, the problem of planning the path of a polygonal rigid object among segment obstacles has been considered. One of the main limitation of the proposed approach is that the computing time increases exponentially with respect to the number of degree of freedom of the object.

References

[1] Boissonnat, J.D., Faverjon, B. and Merlet, J.P., *Techniques de la robotique; perception et planification*, 2, Hermes, 1988.

[2] Brooks, R.A. and Lozano-Pérez, T.A., A subdivision algorithm in configuration space for findpath with rotation; *IEEE Trans. on Sys. Man and Cyber.* 15(2), 1985, 224-233.

[3] Deo, N., *Graphs theory with applications to engineering and computer science*, Prentice-Hall, 1974.

[4] Dijkstra, E.W.,. A note on two problems in connection with graphs, *Numerische Math,* 1, 1959, 269-271.

[5] O. Habert, H. Bullier, A. Pruski, "Distance computing between general shape pre-processed obstacles and general segments-based robot", *IEEE/RSJ International conference on intelligent robots and systems*, IROS, Grenoble, France, 1997.

[6] Jaulin, L. and Walter, E., Set inversion via interval analysis for nonlinear bounded-error estimation, *Automatica*, 29(4), 1993, 1053-1064.

[7] Jaulin, L. and Walter, E., Guaranteed tuning, with application to robust control and motion planning, *Automatica*, 32(8), 1996, 1217-1221.

[8] Jaulin, L. and Godon, A., Motion planning using interval analysis. *MISC'99 Workshop on Application of Interval Analysis to System and Control*, Girona, 24-26 février 1999.

[9] Kieffer M., Jaulin, L., Walter, E. and Meizel, D., Robust autonomous robot localization using interval analysis, *Reliable Computing*, 1999.

[10] Khatib, O., Real-time obstacle avoidance for manipulators and mobile robots, *International Journal of Robotics Research*, 5(1), 1986, 90-98.

[11] Koditschek, D.E., Exact robot navigation by means of potential functions: some topological considerations, *Proceedings of the IEEE International Conference on Robotics and Automation*, Raleigh, NC, 1987, 1-6.

[12] Laumond, J.P., Feasible trajectories for mobile robot with kinematic and environment constraints, *Proc. of Intelligent Autonomous Systems*, Amsterdam, 1986.

[13] Lozano-Pérez, T. and Wesley, M., An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10), 1979, 560-570.

[14] Lozano-Pérez, T., Automatic planning of manipulator transfer movements, *IEEE Trans. on SMC*, 11(10), 1981, 681-698.

[15] Lozano-Pérez, T., Spatial planning: a configuration space approach, *IEEE Trans. on Computers*, 32(2), 1983.

[16] Moore, R.E., *Methods and Applications of Interval Analysis*, SIAM, Philadelphia, 1979.

[17] Nilsson, N.J., A mobile automaton: an application of artificial intelligence techniques, *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, Washington D.C., 1969, 509-520.

[18] O'Dunlaing, C. and Yap, C.K., A retraction method for planning the motion of a disc, *Journal of Algorithms*, 6, 1982, 104-111.

[19] Piazzi, A. and Visioli, A. Global minimum-time trajectory planning of mechanical manipulators using interval analysis, *International Journal of Control,* 71(4), 1998, 631-652.

[20] Pruski, A., Multivalue codes: application to autonomous robots. *Robotic and factories of the future*. Norfolk, USA, 1990.

Software: All algorithms have been implemented with Borland-C++ Builder 3.0 to solve the test-case. The source programs with the associated interval libraries are available on request. A Windows program (.exe) associated with the example treated in Section 4 can be found at ://www.istia.univ-angers.fr/~jaulin/graphdemo.html